# 6. Remote Attestation

Ao Sakurai

2025年度セキュリティキャンプ全国大会
L3 – TEEビルド&スクラップゼミ

# 本セクションの目標

- SGXにおいて最も難解であるRemote Attestation（RA）の内、新型であるDCAP-RAと呼ばれるものついて詳細に理解する。

- DCAP-RA特有のRAインフラ設計の自由度により発生する悩ましい問題についても議論する。

- なお、自著記事[1]に沿って本スライドを作成しているため、各説明の根拠は上記記事内でのリファレンスを参照

# 用語集

| 用語 | 説明 |
| --- | --- |
| Attestationキー | 主にRemote Attestationにおいて非常に重要な役割を果たす鍵で、Provisioningにより生成される。<br>その実はEPIDメンバ秘密鍵。 |
| Intel Key Generation Facility (iKGF) | RPK（後述）やEPID関連鍵等、様々な鍵を作成しストアする、Intelの鍵作成管理施設。<br>インターネットからは接続できない場所に隔離され、強固に保護されている。 |

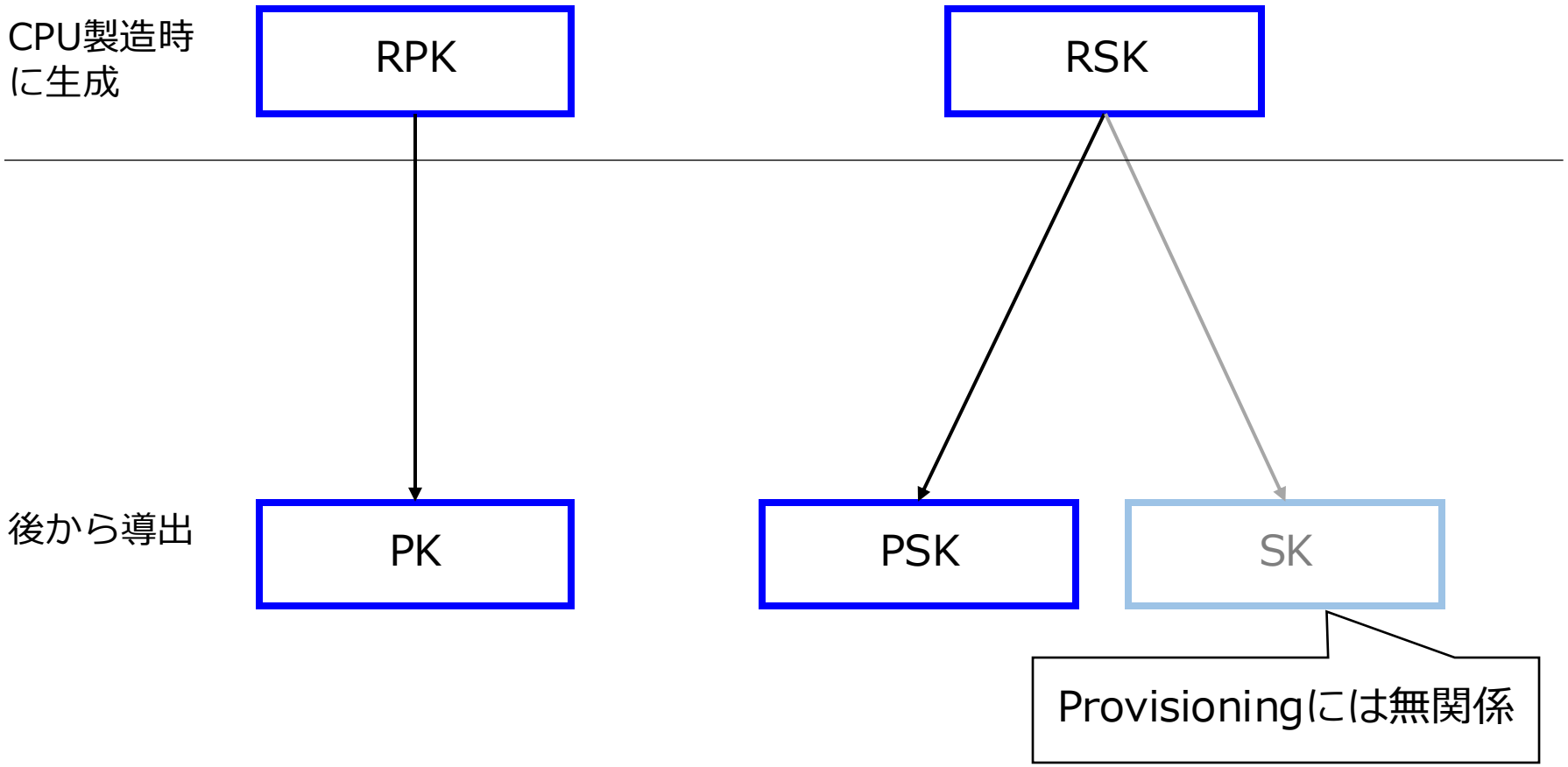
# SGXにおけるHW鍵及びそれに由来する鍵（1/2）

- **Intelによる命名**が**非常に紛らわしい**が、以下のように整理する事が出来る

| 鍵名 | 概要 |
|---|---|
| Root Provisioning Key（RPK; またはProvisioning Secret） | CPU製造時に各CPUのe-fuseに焼き付けられる秘密情報。**この値はIntel側もiKGFで管理・保持している**。 |
| Root Seal Key（RSK; またはSeal Secret） | CPU製造時に各CPU内で乱数的に生成され、e-fuseに格納される秘密情報。**この値はIntel側も保持・把握していない**。 |
| Provisioning Key（PK） | RPKから導出されるプロビジョニング鍵。Provisioningの手続きで使用される。 |
| Provisioning Seal Key（PSK） | RSKから導出される、Provisioning手続き上で必要なシーリングを行う為のシーリング鍵。 |

（参考）Seal Key：通常のシーリングに使用される鍵。これもRSKからポリシ（MRENCLAVE、MRSIGNER）に応じて生成されるが、PSKとは違いOWNEREPOCHという値を有する[2]等の違いがある。

# SGXにおけるHW鍵及びそれに由来する鍵（2/2）




※厳密には、PSKやSKの導出の際、RPKを入力として導出しているらしいSGXマスター導出鍵を使用しているが、この図では省略している。

# Remote Attestationの概要（1/7）

- SGXマシン上のEnclaveの機能を使用したい**リモートのユーザ**が、本当にその**マシン**や**Enclave**が**信頼可能であるか**を**検証**するためのプロトコル

- RAにはEPID-RAとDCAP-RA（本章で説明）の2つが存在し、EPID-RAの方が先発（より古い）である

- SGXプログラミングにおいては**特に難易度が苛烈**であり、このRAさえ実装できれば、SGXSDKで提供されている機能を利用してのSGX関連の実装で**他に恐れるものは無くなる**レベル

# Remote Attestationの概要（2/7）

- 旧式のEPID-RAにおいては、**SGXマシン側**の事を**ISV**（Independent Software Vendor）と呼ぶ事が多かった

- 一方、ISVのSGX機能を利用する**リモートユーザ**（**非SGX側**）は**SP**（Service Provider）と呼ぶ事が多かった

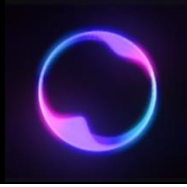
- 何故このような命名であるかは、後のセクションで解説する

# Remote Attestationの概要（3/7）

- 現在では、以下のように呼称する事がTEE全般で一般的である：
  - 検証対象のTEE（SGXサーバ）：**Attester**
  - リモートからTEEを利用したい主体：**Relying Party（RP）**
  - Attesterが提出した証拠を検証する主体：**Verifier**

- 誰が何を担当するかは、そのシステムやインフラの設計次第
  - 例えば、Relying PartyがVerifierを兼ねる場合がある

# Remote Attestationの概要（4/7）

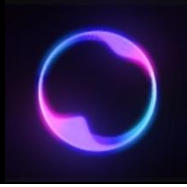


- RAは実装は非常に面倒臭いが、その根源的な目的自体は単純である：

**[RP・Attester]** RA後にTLS用の**セッション鍵を交換する**（LA同様）

**[RP]** AttesterのCPUとEnclaveの**真正性や同一性をリモートから検証する**

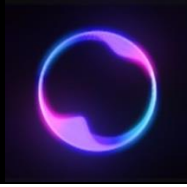
# Remote Attestationの概要（5/7）



- LAでは、相手のマシンの信頼性の根拠として、「**自身と同じマシン上で動作している**」という事をレポートキーの同一性を通して使用していたが、RAでは**リモート**なのでこれは不可能

- 代わりに、**Quoting Enclave**（QE3）がそのEnclaveと**LA**を行った上で、**プロビジョニング**で配備された**Attestationキー**でそのEnclaveの**REPORTに署名**し、**QUOTE構造体**を生成する

# Remote Attestationの概要（6/7）

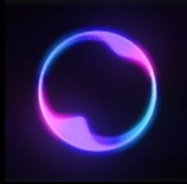


- その後、RPはAttesterから受け取った**QUOTE**を、**検証者**である**Verifier**に送信する

- **Quote署名**に対する、Verifierの持つ**証明書チェーン**による検証や、QUOTE内のREPORT内に存在する**CPUSVN**等から、そのマシンが**信頼可能であるかをVerifierに判定してもらう**

- RPはその判定結果である**アテステーション応答**（**RA Report**）から、リモートのEnclaveが**信頼可能であるかを判断**し、その後**やり取りを続けるかを決定**する
  - Enclave同一性の検証はQUOTE内のREPORTを参照して実施する

# Remote Attestationの概要（7/7）



- LA同様、RAを進める上でRA受理後の**通信に用いる共通鍵**の交換を**EC-DHKE**を用いて実施する

- 相手のEnclaveが本当に意図する通りのEnclaveであるのかの**同一性検証**は、**予め取得した**AttesterのEnclaveの**MRENCLAVE**を、**QUOTE内のREPORT内**に含まれる**MRENCLAVE**と比較する事で**RPが行う**
  - RAでは、**RP側の環境**は**完全に安全**であるという前提で**脅威モデルが設定される**事に注意

# EPID-RAの終焉（1/2）

- かつてはEPID-RAという旧式が存在したが、何と**2025/4/2**に**完全にサービス終了する**という**死刑宣告**され実際に終焉した
  - 開発用プランに至っては2024/9/29に終了

sgx_mailing@intel.com
To 自分

日本語に翻訳

This is an auto-generated message. Please DO NOT reply.

intel

**End of Life Pre-Warning - Intel® SGX Attestation Service Utilizing Intel® EPID**

Dear Customer,

Intel plans to end of life (EOL) the Intel SGX Attestation Service Utilizing Intel® EPID (IAS for short) April 2, 2025. This would include all active API versions. Intel also plans to limit access to the IAS Development (DEV) environment after September 29, 2024.

**Your Action May be Required**

Active consumers of IAS should assess their attestation needs and work to migrate to one of several solution options. Intel-offered attestation alternatives include:

- Intel® Trust Authority – a Zero Trust attestation SaaS
- Intel SGX DCAP with ECDSA-based attestation

Intel will provide periodic reminders as the EOL timeframe approaches.

# EPID-RAの終焉（2/2）

- そもそも、第3世代以降Xeon-SPのような**Scalable-SGX**では、デフォルトで**EPID-RAには非対応**である

- よって、現在ではEPID-RAと並行して存在していたRA方式である、**DCAP-RA**が次世代RA方式として君臨している

# DCAP-RA（1/2）

- その名の通り、**DCAP（DataCenter Attestation Primitives）**というライブラリで提供されている方式のRA
  - ECDSA-RA[4]やThird Party RA[3]という表現を取られる事も多い

- データセンターと名前についているのが示す通り、**サーバ上でSGXを使用**する際に利用できる**汎用的なRA**として生み出されたのが元々の成り立ちである
  - 後のセクションでも述べる通り、EPID-RAをメインとするレガシーSGX（Scalable-SGXより前のSGX）はSGXをクライアント側に置くことを想定して設計されている

# DCAP-RA（2/2）

- 具体的には、以下のような環境を想定して提供され始めたのがDCAP-RAである：
  - イントラネット完結環境のように、**インターネットにアクセス不能な環境**
  - 検証対象Enclave・マシンを信頼するかの判断を**第三者（つまりEPID-RAにおけるIntel）に任せる事を嫌う主体**
  - P2Pネットワークのように、**大規模に分散して動作**するモデルである等の理由により、**単一の検証ポイント（EPID-RAにおけるIntel公式Verifierサーバであ**るIAS相当）**のみで運用する**事が**最適な選択肢でない**シナリオ
  - **EPIDが持つプライバシー特性に反する**か、特にそれを**必要としない**ようなユースケース。特に**秘密計算モデル**としてSGXを使用する場合、サーバのプライバシー（Quoteによるサーバの判別不可能性）を保つモチベーションは**限りなく少ないため**、**サーバ運用でのSGXの大部分がこれに当てはまる**

# DCAP-RAで使用されるAE（1/2）

- **Third-party Quoting Enclave（QE3）**
  DCAP-RAにおいて使用されるQE。”3”は恐らく「Third-Party RA」という呼称から。EPID-RAにおけるQEと異なり、**一般的なECDSAキーペア**を**Attestationキー**として**Quoteの生成**を行う。
  デフォルトではIntel製のものを使用するが、自作QE3の使用も可

- **Provisioning Certification Enclave（PCE）**
  Attestationキーの信頼性確保のために、**PCK**という鍵を用いて署名を行うAE。具体的には、Attestation公開鍵に紐づくハッシュ値をReport Dataに内包したQE3 REPORTにPCK秘密鍵で署名し、**AK Cert**と呼ばれる証明書を生成する。
  PCK自体は**PCK Cert**という、ルートにIntelが君臨する証明書で信頼性が保証される

# DCAP-RAで使用されるAE（2/2）

- **Quote Verification Enclave（QvE）**
  DCAP-RAにおいてQuoteの検証を行うために使用されるAE。
  EPID-RAにおけるIAS相当。主にIntel署名のものが使用されるが、
  QE3同様頑張れば自作のものも使用できる

# EPID-RAとの相違点（1/4）

- Quoteの形式が異なる。DCAP-RAにおいては、**sgx_quote3_t型**として定義されたデータ構造の形を取る[5]。また、Quoteの生成にはQEではなく**QE3**を用いる

- EPID署名は使用しない。使用する署名方式は任意であるが、Intel製QE3では**NIST P-256コンテキストの256bit ECDSA**キーペアを用いている

- Attestation秘密鍵は**QE3内以外で復号される事はない**。EPID-RAのようにPvEとQEで横断的にアンシーリングしないため、その保存には**PSKを用いない通常のシーリング**が使われる

# EPID-RAとの相違点（2/4）

- 前述のサーバ運用SGXにおけるRAの要請から、Quoteを検証する主体が定められていない。つまり、原則としてIASのような**中央集権的なQuote検証者が存在しない**
  - 後述のTrust Authorityを除く

- 製品版Enclaveの利用において、完全に**Intelへのライセンス登録が不要**。EPID-RAの場合、IASのAPI側の問題でライセンス登録が必須であるという制約が存在した

- Quote検証をするには、**検証者が自前でQvEを立てて行う**。Enclave外で完結するQvLを用いる選択肢もある

# EPID-RAとの相違点（3/4）

- **QE3やQvEの動作定義**は**任意に変更可能**。同時に、これらにIntelによる署名がついていなくても良い

- Quote署名の信頼性を担保するために、PCEにてAttestation公開鍵に対し**PCK秘密鍵**で署名を打つ。さらに、PCK公開鍵に対する、Intelがルートの証明書チェーン（PCK Certチェーン）で担保する

- IAS以外の主体がQuoteを検証するため、それを補助するための付属情報（**コラテラル**）が用意されている。
PCK Certチェーン、TCB Info(*)、QE3同一性情報(*)、PCK Cert失効リスト（PCK CRL）(*)、ルートCA CRLがあり、
(*)で印をつけた3つの要素には発行者証明書チェーンがある

# EPID-RAとの相違点（4/4）

| 相違点 | EPID-RA | DCAP-RA |
|---|---|---|
| **Quoteの形式** | sgx_quote_t | sgx_quote3_t |
| **Quoteを生成するAE** | QE | QE3 |
| **Quote署名方式** | EPID署名 | NIST P-256 ECDSA署名 |
| **Attestation秘密鍵が復号され得る場所** | QEとPvE | QE3のみ |
| **Quote検証者** | IAS | 例外を除いて不定（自由） |
| **製品版Enclaveのライセンス申請** | 必要 | 不要 |
| **Quote生成と検証のロジック** | Intelが用意したもの（QE、IAS）で固定 | 自由（自前のQE3やQvE・QvL等） |
| **Quote署名の信頼性担保** | IASによるEPID署名の直接検証及び結果へのIntel署名 | Intel発行のPCK Certチェーン、PCE |
| **Quote検証に必要な情報** | 特に無し（強いて言えばSPIDとIASサブスクリプションキー） | コラテラル |

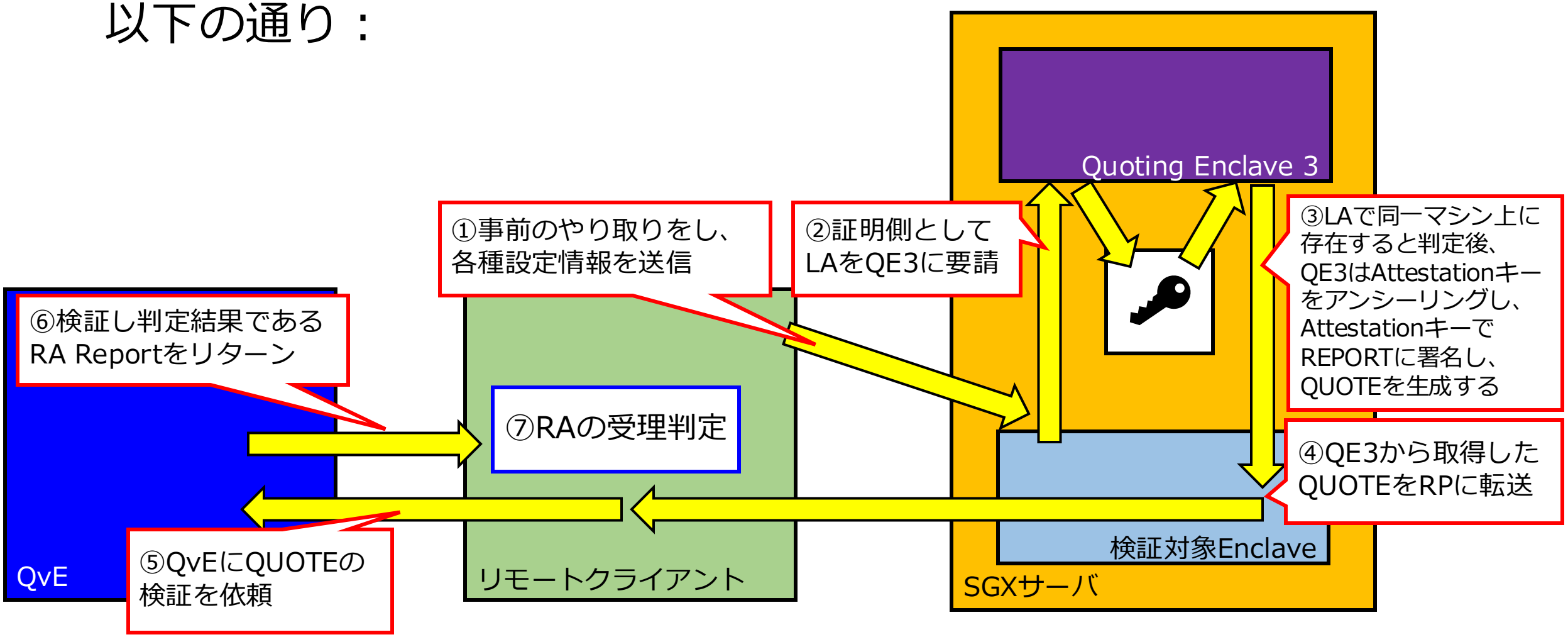
# 各主体の呼称

- EPID-RAの場合と異なり、**SGXマシン側**を**サーバ**、**非SGXのリモートユーザ**を**クライアント**と呼ぶ
  - DCAPがサーバ運用を想定しており、それにより後のセクションで説明するISVやSPの意図する所から乖離しているため
  - あるいは、より一般的な呼び方として、前述の通りAttesterやRPという表現をしてももちろん良い

- ただし、DCAPライブラリの実装においてはISVという表現が多用されていたりするため、**伝統的に使用**するという意味で**ISVやSPという表現も誤りではない**

# DCAP-RAの概要図

- DCAP-RAのQuote生成から検証、RA受理判断までの概要図は以下の通り：



# 自由度に伴う3つの議論（1/2）

- DCAP-RAは**汎用性を重視**しているため、RAの**各処理の責任の所在**が**明確に定められていない**という性質がある

- 一見この自由度は魅力的に見えなくもないが、以下の3つのような設計上の悩ましい問題をもたらす原因となっている
  - EPID-RAにおける**IAS相当の処理（Quote検証）**を**誰が行う**か？
  - Quoteの検証に使用する付属情報**（コラテラル）を誰が取得**するか？
  - Quote検証者に**Quoteを送信するのは誰**か？

- 特にQuote検証者を誰にするかという問題は悩ましい
  - SGXマシンで行うとセキュリティ的にかなり怪しいが、クライアントが担当するには負担が大きく、第三者検証サーバを立てるとそれ自体の信頼性の問題がある

# 自由度に伴う3つの議論（2/2）

- 本スライドでは、原則として以下のようにこの3つの自由度を固定して説明を進める
  - **Quote検証者→クライアントが完全に信頼**している**専用の検証サーバ**。例えばクライアントが自前で用意した検証サーバ。内部で確実にQvEまたはQvLを動作させていることを確信できるものとする
  - **コラテラル取得者→Quote生成時**は**SGXサーバ**。**Quote検証時**は**Quote検証サーバ**
  - **QuoteをQuote検証者に送信する主体→クライアント**。SGXサーバからQuoteを受け取り、それをQuote検証サーバに転送して検証を要請する
- ただし、適宜上記以外の場合の説明も挟む事もある

# DCAP-RAにおけるトラストチェーン（1/3）

- DCAP-RAの信頼性を保証するための重要な要素である、**DCAP-RAにおけるトラストチェーン**についてここで説明する

- DCAP-RAでは、Quote署名はECDSA、検証機関は原則Intel以外、そしてQE3やQvEすらIntel製でなくても良いため、このままでは**Quoteの信頼性に不安が残る**
  - 謎のQE3が生成した野良のECDSA署名で生成された怪しいQuoteを、ちゃんと検証しているかすら確信できないQvEに渡して得たQuote検証結果は信頼できるのか？

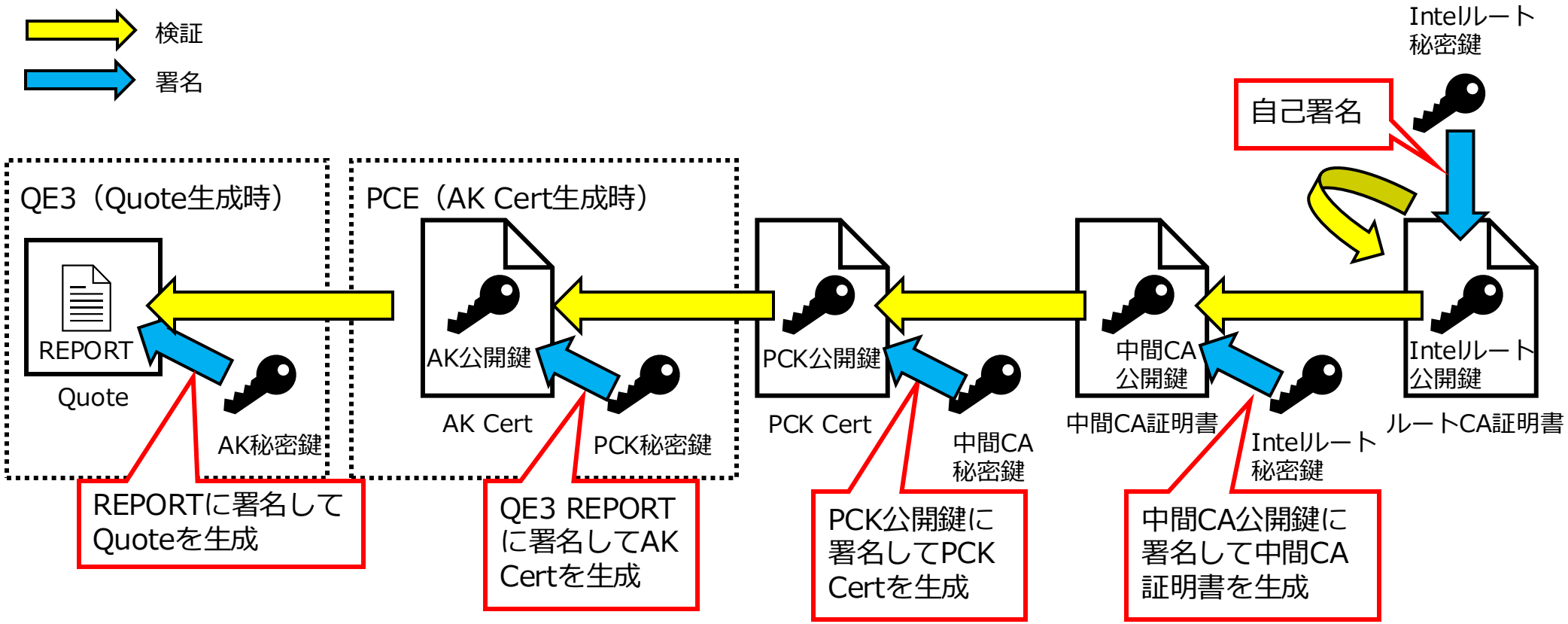
- EPID-RAの場合は、Intelが管理するEPIDグループに基づいており、かつIASが直接検証していたのでこのような問題は無かった

# DCAP-RAにおけるトラストチェーン（2/3）

- この問題を解決するため、DCAP-RAではQuoteやAttestationキーを、Intelをルートとした以下のようなトラストチェーンにより保証する方式を採用している：
  - QE3は検証対象の**REPORT**に**Attestation秘密鍵で署名**し**Quoteを生成**
  - その検証のためのAttestation公開鍵の信頼性を保証するために、Intel製AEであるPCEが**PCK秘密鍵**でAttestation公開鍵（に紐づくデータである**QE3 REPORT**。詳細は後述）に**署名**し、**AK Certを生成**
  - 今度はPCK公開鍵の信頼性を保証するため、**Intel**は**PCK公開鍵に対する証明書**である**PCK Cert**を保持し提供する
  - PCK Certは1つ上の**中間CA秘密鍵で署名**されており、その**中間CA証明書**（中間CA公開鍵）は**IntelのルートCA証明書で検証できる**

# DCAP-RAにおけるトラストチェーン（3/3）

- DCAP-RAのトラストチェーンを図示すると以下の通り：
  - PCK Cert及び上位の証明書（PCK Certチェーン）はコラテラルとして提供され、QuoteはRA本処理で生成されるため、その**中間**の**Attestationキー**と**AK Cert**を**生成**するのが**プロビジョニングの主な仕事**



# コラテラル

- **コラテラル**：DCAP-RAにおいて**Quoteの生成や検証**の際に使用される**付属情報**
  - **PCK Certチェーン**：PCK CertからIntelルートCA証明書までの証明書チェーン
  - **TCB Info**：マシンのSVNとそれに対応するマシンのセキュリティ状態（TCBステータス、アドバイザリID等）のリスト
  - **QE3同一性情報**：Intel製QE3の同一性情報（MRENCLAVE等）
  - **PCK CRL**：PCK Certに対する失効リスト
  - **ルートCA CRL**：ルートCA証明書に対する失効リスト
- PCK CRL、TCB Info、QE3同一性情報にはそれぞれに対するIntelをルートとした発行者証明書チェーンも配布される

# コラテラルサービス（1/2）

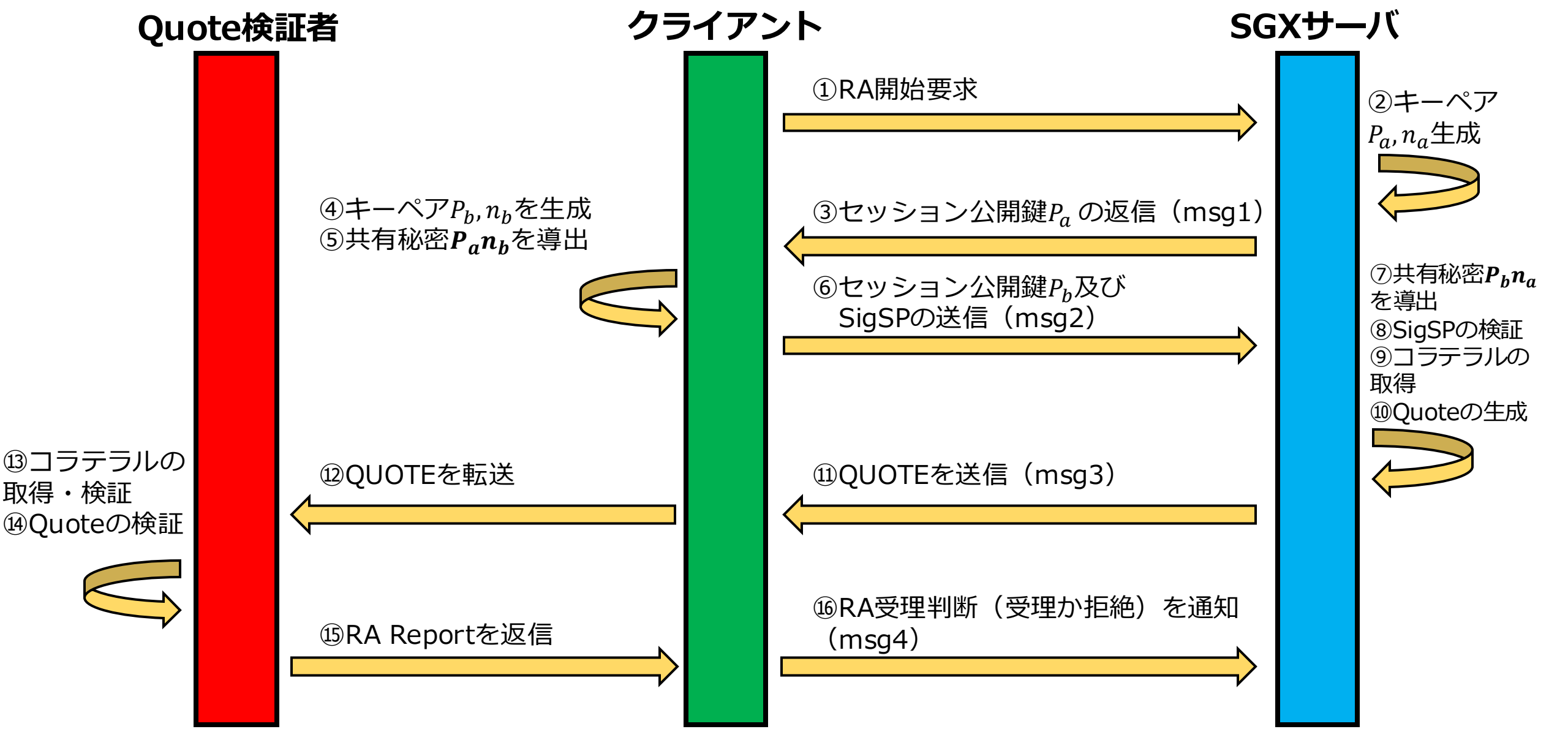
- 文字通り、コラテラルを配布してくれるサービスの事

- Intelは、**Provisioning Certification Service（PCS）**で各種コラテラルを配布している
  - 以下のリンク先でAPI利用申請が可能
    https://api.portal.trustedservices.intel.com/provisioning-certification
  - この申請も基本的にベアメタル環境を一からセットアップする際にのみ必要な作業（Azure VM利用時等は不要）

# コラテラルサービス（2/2）

- しかし、毎回PCSにリモートアクセスしていては余計な通信時間がかかってしまうし、規約で禁止されている[16]
  - DCAP-RAを**イントラ完結で運用したい要求**に**応える事もできない**
- そこで、PCSから取得したコラテラルをキャッシュしておく**PCCS（Provisioning Certificate Caching Service）**がDCAPライブラリとして提供されており利用可能
  - RA実施前にPCCSにキャッシュしてそれを使う事で、イントラ環境でのDCAP-RAも実現可能になる
- ちなみに、**Azure VM**でSGXを使用する場合は、Azureが提供する**THIM（Trusted Hardware Identity Management）**をPCCSとして使用可能である

# DCAP-RA本処理の全体フロー



# DCAP-RAプロビジョニング

# DCAP-RAにおけるプロビジョニング

- EPID-RAにおいても存在したプロビジョニングであるが、
  DCAP-RAにおいても形は違うが存在する

- DCAP-RAのプロビジョニングは、RA本処理に向けて
  以下の4つの処理を行うものである：
  - Intelへのプラットフォームの登録（マルチソケット環境のみ）
  - QE3内でのAttestationキーペアの生成
  - AK Cert生成のためのPCKの作成
  - QE3で生成されたAttestation公開鍵に対する、PCE内でのAK Certの生成

# プラットフォーム登録（1/5）

- Xeonプロセッサでは、1つのマシンが2つ以上のCPUを持つ「**マルチソケット**」構成を取る事ができる
  - CPUが増える分並列処理等パフォーマンス面の恩恵が大きい

- しかし、SGXにおいては**CPU固有の鍵由来の鍵**を使用する際に、マルチソケット環境というものはそのままでは困った事になる
  - 例：どのCPUのRPKをベースにPCKを導出する？

- これを解決するため、プラットフォームをIntelのサービスに登録し、その後プラットフォームを代表するRPKのような存在である**プラットフォーム鍵**を生成する

# プラットフォーム登録（2/5）

- 当然、単一CPUのみの環境（**シングルソケット環境**）では
  このプラットフォーム登録処理は**不要**

- また、ベアメタルマシンを手に入れて1からセットアップする際に
  走る処理であるため、例えばAzureでデプロイされたVMにおいて
  実行する必要なども無い

- プラットフォーム登録が必要である場合、まずその
  プラットフォームの全てのCPUの情報に基づき、その
  プラットフォームに一意な**128bit鍵（プラットフォーム鍵）**を
  マイクロコードが生成する

# プラットフォーム登録（3/5）

- 次に、実際に**IRS**（Intel Registration Service）への
  プラットフォーム登録が発生するが、その際にプラットフォームは
  **Platform Manifest**というデータ構造をマイクロコードで生成する

- Platform Manifestの内訳は以下の通り[6]：
  - 各CPUパッケージに関する情報
  - 各CPUのPlatform Registration Key（PRK）による署名
  - IRSの公開鍵であるRegistration Server’s Encryption Key（RSEK）で
    暗号化したプラットフォーム鍵

- PRKとRSEKは、いずれも**3072bit RSA暗号鍵**である

# プラットフォーム登録（4/5）

- その後、以下の手続きでIRSへのプラットフォーム登録を行う[6]：

1. マイクロコードにより、そのプラットフォームに紐づく情報である**Platform Manifestを生成**する

2. BIOSを通じてPlatform Manifestを**IRSに送信**する

3. IRSは、Intel署名付きの**PRK Cert**（PRKに対する証明書）を用いて、Platform Manifest内の**PRK署名を確認**する。この検証に成功したらプラットフォーム登録は完了。

# プラットフォーム登録（5/5）

- PRKはマシン初期化時やマイクロコードのSVNが更新されるような TCB Recovery時に更新される

- PRK CertはデフォルトでIntelが全てのPRKに対して有しており、PRKが更新されると必ず対応する新規のPRK Certが生成される

- つまり、**IRSは全てのCPUパッケージに対するPRK Certを有している**事になる

# Attestationキーペアの生成

- プラットフォーム登録以外の**プロビジョニング**は、QE3による検証対象EnclaveとのLAのための、**QE3のTarget Info取得時**に実行される[7]
  - プロビジョニング済みかつ再プロビジョニング不要な際には実行されない

- まずは、**QE3**内で**Attestationキーペアを作成**する
  - Intel製QE3では、NIST P-256曲線を使用したIETF RFC 6090準拠の256bitのECDSA署名鍵であり、QE3に紐づくMRSIGNERポリシのシーリング鍵をベースとしている

- 一方、Attestationキーペアを乱数的に生成する選択肢も用意されている
  - 同一QE3を別々のVMで動かす場合等に最適な選択肢

# PCKの導出（1/2）

- 前述で生成した**Attestation秘密鍵**は**Quoteへの署名**に使われ、**Attestation公開鍵**は**Quoteの検証**に用いられる

- このAttestation公開鍵をトラストチェーンに縛るための証明書が**AK Cert**

- AK Certの生成（Attestation公開鍵+αへの署名）には**PCK（Provisioning Certificate Key）**が使用される

# PCKの導出（2/2）

- PCKは**PK**（究極的にはその導出元である**RPK**）を**ベース**に導出される

- 導出の際は以下の要素についてもベースとしており、PCKはRPKに並びこれらについて一意な鍵となっている
  - PCKを導出する**マシン**
  - 現在のSGXのTCBSVN（つまりは恐らく**CPUSVN**）
  - PCEのISVSVN（**PCESVN**）

- PCKもIntel製QE3のAttestationキーと同じく、NIST P-256曲線を使用したIETF RFC 6090準拠の256bit ECDSA署名鍵である

# PCK Cert（1/4）

- 文字通り、**PCK公開鍵に対する証明書**
  - 中間CAの秘密鍵により署名される
  - その中間CAの公開鍵がIntelのルート秘密鍵で署名される
- QuoteやAK Certと異なり、**Intelによって生成**される
  - IntelはiKGFで全てのマシンのRPKを管理しており、プラットフォーム鍵の場合も登録により把握しているため、PCKを再現する事ができる
- PCK Certのコラテラルサービスからの取得は**Quote生成時**に行われる

# PCK Cert（2/4）

- コラテラルサービスからのPCK Certの取得の際は、**PCEID**と**暗号化されたPPID**という2つの値をクエリとして行う
  - **PPID**：そのマシン及びPCEの同一性に対して一意なID。Platform Provisioning ID。暗号化方式にはデフォルトでは3072bit RSA-OAEPが使用される
  - **PCEID**：PPIDとPCKを導出するのに用いたPCEの同一性に対して一意なID

- セキュリティに関係しないCPUSVNインクリメント時に対してはPCK Certは発行されないため、取り得る全てのPCKのパターンに対してPCK Certが存在するわけではない
  - 存在するPCK Certに合わせるために、PCK生成時には使用するCPUSVNを（そのマシンの最新値以下の範囲で）指定する事ができる

# PCK Cert（3/4）

- PCK CertはX.509証明書の形を取っており、そのX.509拡張領域（Extentions）に以下のSGX関連の情報が格納されている：
  - PPID
  - そのマシンのCPUSVNと使用されたPCEのPCESVN
  - FMSPC（Family-Model-Stepping-Platform-CustomSKU）

- FMSPCは、そのCPUのファミリ、モデル、ステッピング、プラットフォームタイプ、及びカスタマイズSKUを含む値
  - そのCPUやプラットフォームに紐づくバージョンとして機能する

# PCK Cert（4/4）

- PCK Certは検証対象マシンのPCKに対する証明書であるため、PCK CertによるPCKの検証に成功した時点で、その**PCK Cert**は**そのマシンに紐づくデータである**事になる

- つまり、**PCK Certの拡張領域内の情報**をTCB Infoと照合しながら**検証**する事により、**検証対象マシンの安全性を検証**できる事になる
  - このように、PCKに対する証明書としてだけでなく、対象のマシンのセキュリティレベルを表すデータであるため、PCK Certは非常に重要

# AK Certの生成（1/3）

- AttestationキーペアとPCKキーペアの生成が完了したので、最後に**AK Certの生成**を行う

- AK Certの生成（Attestation公開鍵へのPCK秘密鍵による署名）はPCEによって行われるが、AttestationキーはQE3で生成されるため、**改竄を防ぎながらQE3からPCEに配送**する必要がある

- これを実現するために、PCEはQE3に対して**LAを実施**する
  - PCEのTarget Infoを入力とした**QE3 REPORT**をQE3に生成させそれを検証する

# AK Certの生成（2/3）

- このQE3 REPORTは、**Report Data領域**に以下の2つの連結に対するSHA-256ハッシュ値を格納している：
  - Attestation公開鍵
  - QE3認証データ（QE3が指定する追加情報。Intel製QE3ではダミーを挿入しており事実上使用していない）

- このQE3 REPORTに対してPCK秘密鍵で署名したものが**AK Cert**である

# AK Certの生成（3/3）

- QE3 REPORTやAK CertはQuote生成時にQuoteに同梱するために**後ほど再使用**される

- Attestationキーペアについても、**秘密鍵**は**Quoteの署名**に使用され、**公開鍵**はQuoteに同梱され**Quote署名の検証**に使用される

- よって、QE3はメッセージ（平文）としてAttestationキーペア、AAD（追加認証データ）としてQE3 REPORTやAK Certを入力とした、**MRSIGNERポリシのシーリング**を行いファイルシステムにストアする
  - このシーリングはPSKによるものではなく、**通常のシーリング**

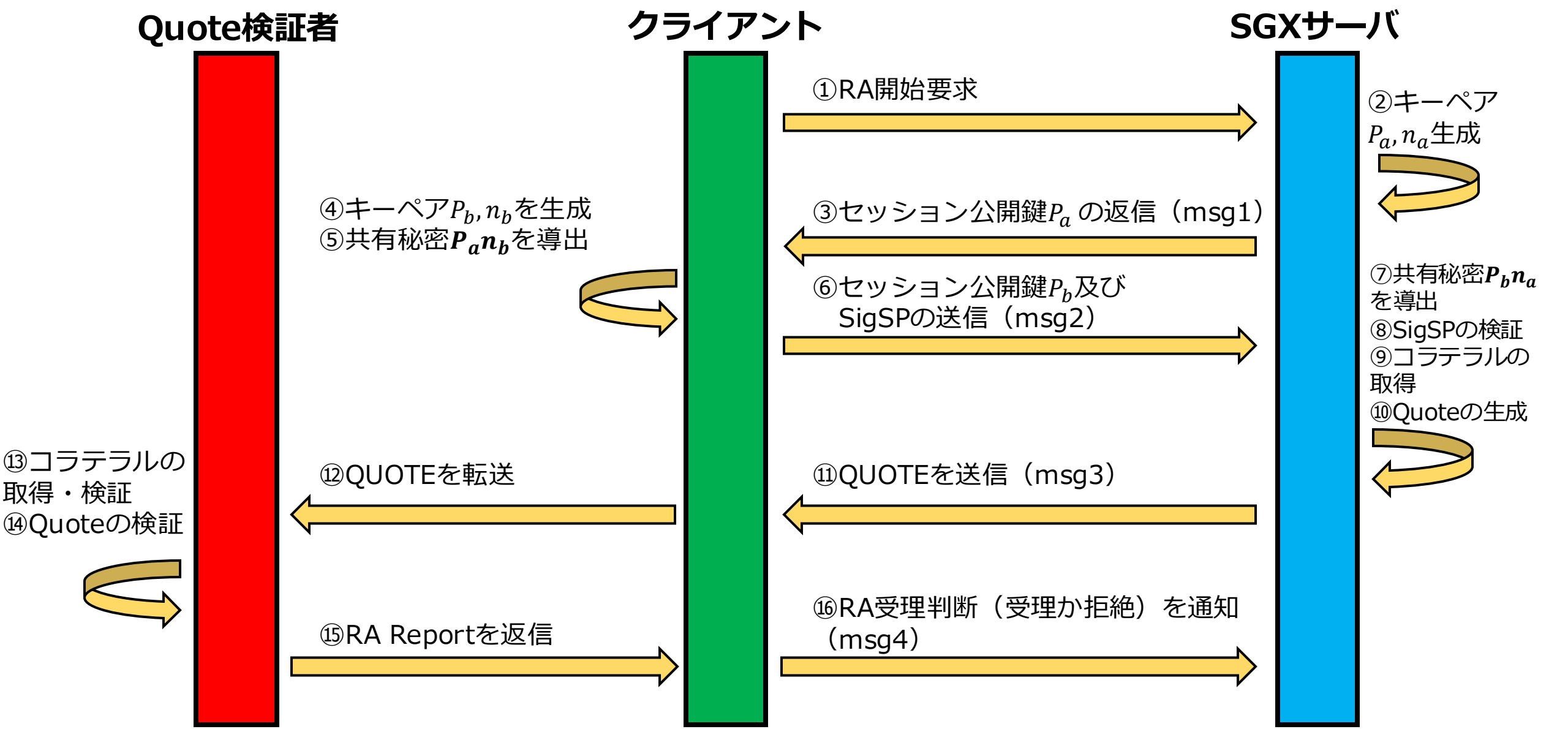
# DCAP-RA本処理

# 取り扱うDCAP-RA本処理の前提

- かつてのEPID-RAに倣い、msg1～msg4をやり取りする事により鍵交換とRAを同時に進行させるものについて考える
  - ただし、DCAP-RAではmsg1～4のラベル付けは公式にはされていない

- SGXのRAが想定する脅威モデルに従い、クライアントの環境は完全に信頼可能であるとする

# DCAP-RA本処理の全体フロー



# RA開始要求

- EPID-RA同様、クライアントはサーバにRAを開始するためのリクエストを送信する
  - 単にSGXサーバが構える特定のポートやURL等にアクセス（GETリクエスト等）するだけで良い

- 直後の処理でEnclaveを使用するため、このタイミングまでにSGXサーバは検証対象Enclaveを起動しておくと良い

# msg1の生成

- SGXサーバは、Enclave内でEC-DHKEのためのセッションキーペア$P_a, n_a$を生成する
  - sgx_ecc256_compute_shared_dhkey()によりNIST P-256の256bit ECDSA鍵を生成するのが楽

- EPID-RAと異なり、**RAコンテキスト**を始めとした相手毎の各種情報は**自動生成及び管理されない**ため、**自前で作成し管理する**必要がある
  - 実装例はこちらのra_session_tを参照

- SGXサーバ側セッション鍵$P_a$と、相手クライアントに割り当てたRAコンテキスト値をmsg1として返信する

# msg2の生成

- msg1を受信後、クライアント側セッションキーペア$P_b, n_b$を生成する
- そして、EPID-RAの場合同様に共有秘密$G_{ab}$を生成し、その$x$成分である$G_{ab_x}$を用いて鍵導出鍵KDKを作成する
- さらに、必要であればEPID-RA同様SigSPも生成する
  - 後にReport Dataに両者のセッション公開鍵に紐づくハッシュを同梱するため、実装に自由度があるDCAP-RAではSigSPは必須ではない
- RAコンテキスト、クライアント側公開鍵$P_b$、そして使う場合はSigSPをmsg2として同梱しSGXサーバに送信する

# msg3の生成 - 鍵交換関連処理

- msg2を受信後、Enclave内で共有秘密$G_{ab}$を生成し、KDK、VK、SK、MKを生成する
  - EPID-RAではsgx_ra_proc_msg2()により自動的にされていた処理
  - SMKは使用しないので作成不要
- SigSPを渡されている場合は、Enclaveにハードコーディングされた署名検証用公開鍵を用いてその検証も行う
- 上記の実装例はこちらのecall_process_session_keys()関数を参照

# msg3の生成 - QE3 Target Infoの取得

- ここからは、提供されているAPIレベルでの**ハイレベル観点**での説明をまず行う
  - Quoteの生成や検証の、ローレベル観点での詳細な仕組みについては本セクション内で後から解説する

- sgx_qe_get_target_info()関数を呼び出す事により、検証対象EnclaveのREPORT作成のための**QE3のTarget Info**を取得する
  - 前述の通り、このタイミングで必要であればプロビジョニングが行われる

# msg3の生成 - 検証対象EnclaveのREPORT生成

- QE3 Target Infoを引数として渡しながら**sgx_create_report()**関数を呼び出し、検証対象EnclaveのREPORTの生成を行う

- このREPORTのReport Dataには、中間者攻撃対策としてEPID-RAの場合と同様、$P_a, P_b, VK$の連結に対するSHA-256ハッシュ値を上位32バイトに内包させる必要がある

- よって、実際にはECALLでEnclaveに入り、Enclave内で保持している$P_a, P_b, VK$からハッシュ値を導出し、それをECALL時に外から持ち込んだTarget Infoと共にsgx_create_report()に渡す

# msg3の生成 - Quoteの生成

- REPORTを生成したら、**sgx_qe_get_quote_size()**により生成されるQuoteサイズを取得する
- その後、上記サイズ分のQuote用バッファを確保し、**sgx_qe_get_quote()**を呼び出す事でQuoteを取得できる
- その**Quote**を**msg3**としてクライアントに送信する
- DCAP-RAにおけるQuote構造体の詳細な仕様については後述

# Quoteの検証

- クライアントは、SGXサーバから受信したmsg3（Quote）をそのままQuote検証サーバに転送する

- Quote検証サーバは、Quoteを検証するために**QvE**と**QvL**（Quote Verification Library）のいずれかを使用できる

# QvEとQvL

- **QvE**を用いる場合、終始**Enclave（QvE）内**で検証処理が進む
  - QvEが動作している事を確信できる環境であれば、**正しい検証処理が行われている**事が（余程の事が無い限り）**保証**される
  - 検証対象Enclaveの載るマシンでQvEも動かす場合は、検証対象EnclaveからQvEをLAで縛る事もできる

- **QvL**は終始**Untrusted領域（Enclave外）**で完結する
  - よって、少なからず**信頼性の観点ではQvEには劣る**
  - しかし、QvLを用いればQuote検証に**SGX環境が不要**であるため、**汎用性**の観点では**遥かに上**
  - クライアントが自身で検証する等、**完全に信頼可能な場**で**Quoteを検証**する場合に特に有効な選択肢

# Quote検証の準備（1/3）

- Quote検証サーバは、Quoteの検証やその結果受け取りのために以下に列挙するものを揃える：
  - 検証対象**Quote**（クライアントから転送される）
  - **コラテラル期限切れステータスフラグ**変数（uint32_t型で宣言すれば良い）
  - 検証開始時点での現在時刻（タイムスタンプ）
  - Quote検証結果**補足情報（後述）**
  - **QvE REPORT**情報（**後述**）
  - QvE検証用nonce（検証対象EnclaveのマシンでQvEを動かす場合）

# Quote検証の準備（2/3）

- **コラテラル期限切れステータス**：文字通り、検証に使用したコラテラルが期限切れを起こしているかのフラグ
  - **期限切れ時は1となる**が、期限切れを起こしていても**致命的なエラーとは見なされない**。期限切れを受理するかは**クライアントの判断次第**

- **Quote検証結果補足情報**：Quote検証結果についての詳細や、RA受理判断に利用可能な情報を内包する構造体。
  - 型はsgx_ql_qv_supplemental_t。構成の詳細は後述
  - 英語ではSupplemental Data。以下、「**補足情報**」とも言及する
  - 受け取るかは任意であり、不要な場合は後述のtee_verify_quote()の補足情報部分の引数をNULLとする

# Quote検証の準備（3/3）

- **QvE REPORT情報**：QvE検証用nonce、検証対象Enclaveの Target Info、QvEのREPORTの3つを格納する構造体
  - 前者2つはtee_verify_quote()への入力として機能し、QvE REPORTは同API内で前者2つをベースに生成されるものである
  - **QvE REPORT情報**を**tee_verify_quote()**に**渡すか否か**により、**QvE**（渡した場合）と**QvL**（渡さない場合）の**どちらを使用するかが決定**される
- 補足情報はtee_get_supplemental_data_version_and_size()で**補足情報**の**バージョン**と**サイズ**を取得しておく
- QvEを用いる場合は、sgx_qv_set_enclave_load_policy()で**QvEのロードポリシー**（QvEを都度生成するかプロセスセッション内で起動したままにするか等）を指定する

# Quoteの検証（1/2）

- 準備が整ったら、**tee_verify_quote()**関数を呼び出す事で**Quoteの検証を実施**する
  - 前述の通り、ここでQvE REPORT情報を渡すとQvEを用いた検証となる

- このAPIによる検証の結果として以下の情報が返される：
  - コラテラル期限切れステータス（期限切れ時は1、それ以外は0）
  - Quote検証結果（いわゆるOKやGROUP_OUT_OF_DATE相当の、**RA結果ステータス**）
  - 補足情報（非NULLバッファを渡した場合のみ）
  - QvE REPORT（QvEを用いた場合のみ、QvE REPORT情報に格納される）

# Quoteの検証（2/2）

- QvEを検証対象Enclaveの載るマシンで動作させている場合は、sgx_tvl_verify_qve_report_and_identity()により、受け取ったQvE REPORTを用いて**検証対象EnclaveによるQvEとのLA**を行う事もできる
  - このLAについてのセキュリティ上の議論については後述

- Quote検証結果として、クライアントに例えば以下の情報を返却する（フォーマットは定められていない）：
  - 検証したQuote
  - Quote検証結果ステータス
  - コラテラル期限切れステータス
  - 補足情報

# RA受理判断の実行・msg4の送信（1/3）

- Quote検証結果一式を受信したら、クライアントはそれらを確認し検証対象マシン及びEnclaveを信頼するかを決定する

- Quote検証結果ステータスは、以下のようにして判断する

| Quote検証結果ステータス | 判断 |
|---|---|
| SGX_QL_QV_RESULT_OK | **原則として受理で良い**。ただし、コラテラルが期限切れであるのを不許可とする場合は、コラテラル期限切れステータスに応じて棄却する |
| SGX_QL_QV_RESULT_INVALID_SIGNATURE、SGX_QL_QV_RESULT_REVOKED、SGX_QL_QV_RESULT_UNSPECIFIED | 署名が無効であるか失効している、あるいは不明なエラーが発生しているため、**絶対に受理してはならない** |
| SGX_QL_QV_RESULT_CONFIG_NEEDED、SGX_QL_QV_RESULT_OUT_OF_DATE、SGX_QL_QV_RESULT_OUT_OF_DATE_CONFIG_NEEDED、SGX_QL_QV_RESULT_SW_HARDENING_NEEDED、SGX_QL_QV_RESULT_CONFIG_AND_SW_HARDENING_NEEDED | 原則として何らかの脆弱性や設定上の不備を抱えている。補足情報内のアドバイザリID等も参照しつつ、脅威レベルに応じて**クライアントが各自受理するかを判断する** |

# RA受理判断の実行・msg4の送信（2/3）

- 次に、EPID-RAと同様、予め控えておいた**期待する同一性情報**と、**Quote内の同一性情報**を**比較**する
  - 主に**MRENCLAVE**、**MRSIGNER**、ISVSVN、ISV Prod ID
  - 実装例としてはこちらのverify_enclave()関数を参照

- Quote内のREPORTの**Report Data**についても検証する
  - まずはKDKからVKを生成する（生成方法はSGXサーバと同じ）
  - そして、$\boldsymbol{P_a, P_b, VK}$**の連結に対するSHA-256ハッシュ値**を導出し、それが**Report Dataの上位32バイトと一致する**かを確認する

# RA受理判断の実行・msg4の送信（3/3）

- 上記までの検証の結果、最終的に**RAを受理するか棄却するか**をクライアントは**判断する**

- その**判断結果**を**msg4**として**SGXサーバに送信**する
  - EPID-RAと同様、msg4のフォーマットは完全に自由
  - 例えばOKかNGかのフラグだけでも良く、あるいは参考情報として判断理由やQuote検証結果補足情報を送っても良い

- RAを受理する場合は、クライアント側でもSKとMKを生成し、好きな方を用いて**SGXサーバのEnclaveとの暗号通信**を行う

# Quote生成ロジック詳説

# 検証対象EnclaveとのLA完了処理

- ここからは、DCAP-RAにおけるQuote生成についての
  非常に詳細な解説を行う

- sgx_qe_get_quote()が呼び出されると、まず**QE3**は**検証対象EnclaveのREPORT**を**レポートキーにより検証**する事で、**LAの完了処理**を行う
  - この処理は定石通りsgx_verify_enclave()を呼び出す事により
    実現されている

# DCAP-RAにおけるQuoteの構造（1/4）

- 前述の通り、DCAP-RAにおけるQuoteはEPID-RAのものとは異なっており、**高レベル観点**（sgx_quote3_tに準じる）では以下のような構造になっている

| メンバ | 説明 |
|---|---|
| **sgx_quote_header_t** header | Quoteヘッダ。詳細は後述 |
| **sgx_report_body_t** report_body | 検証対象Enclaveで生成した、QE3のTarget Infoを入力としたREPORT構造体。注意点として、sgx_report_tではなくsgx_report_body_tであるという、DCAP-RAのQuote特有の仕様がある |
| **uint32_t** signature_data_len | 後ろに続く署名関連情報のデータ長 |
| **uint8_t** signature_data[] | 署名関連情報。詳細は後述 |

# DCAP-RAにおけるQuoteの構造（2/4）

- Quote内のREPORTが**ボディ部分だけである**（レポートキーによるMACが付与されていない）ため、**一見危険**であるようにも見える

- しかし、LAによるREPORT検証後に**REPORTはQE3外に出る事がなく**、かつ**QE3内完結**でsgx_report_tから**ボディ部分を抽出**して**Quote署名を打つ**ため、結局の所**改竄の余地はなく安全**
  - 改竄するのであればQuote内のREPORTボディに対して行う事になるが、言うまでもなくQuote署名の検証で検知できる

- 前述の**QE3 REPORTもREPORTボディの形を取っている**
  - こちらもプロビジョニング後にシーリングされ、Quote生成時にQE3でアンシーリングされQuoteにそのまま格納されるので安全

# DCAP-RAにおけるQuoteの構造（3/4）

- まず、Quoteヘッダについてローレベル観点で以下の表に示す：

| メンバ | 説明 |
|---|---|
| **uint16_t** version | Quote構造体のバージョン |
| **uint16_t** attestationKeyType | QE3により使用されるAttestationキーのタイプ。これが2であれば、本スライドでも前提としているNIST P-256 ECDSAキーペアを使用している事を示している |
| **uint32_t** teeType | TEEタイプ。0x00であればSGX用の、0x81であればTDX用のQuoteである事を表す |
| **uint16_t** qeSvn | そのQuoteの生成に使用されたQE3のISVSVN |
| **uint16_t** pceSvn | そのQuoteの生成に関与したPCEのISVSVN |
| **std::array<uint8_t, 16>** qeVendorId | QE3のベンダID。IntelのQE3の場合は、939A7233F79C4CA9940A0DB3957F0607というIDが割り振られている |
| **std::array<uint8_t, 20>** userData | ユーザ定義のデータ。Intel公式のQE3の場合、暗号化済みのPPIDとPCK Certを紐づけるためのQEIDという値が先頭16バイトに格納されている。 |

# (復習) REPORTボディの構造

- sgx_report_body_tの構造（1/2）

| メンバ | 説明 |
|---|---|
| **sgx_cpu_svn_t** cpu_svn | CPUのセキュリティバージョン番号。TCB Recoveryで更新 |
| **sgx_misc_select_t** misc_select | 将来実装されるかも知れない機能のための予約領域 |
| **uint8_t** reverved1[12] | 将来のための予約領域。現時点ではオールゼロとする |
| **sgx_isvext_prod_id_t** isv_ext_prod_id | ISVに割り当てられた拡張Prod ID。KSS（Key Separation and Sharing）機能を使用する際に使われる[12] |
| **sgx_attributes_t** attributes | Enclaveの権限や属性に関する設定をする値 |
| **sgx_measurement_t** mr_enclave | REPORTを発行したEnclaveのSECSから取得したMRENCLAVE |
| **uint8_t** reserved2[32] | 将来のための予約領域。現時点ではオールゼロとする |
| **sgx_measurement_t** mr_signer | REPORTを発行したEnclaveのSECSから取得したMRSIGNER |
| **uint8_t** reserved3[32] | 将来のための予約領域。現時点ではオールゼロとする |
| **sgx_config_id_t** config_id | Enclave設定のID。KSS機能を使用する際に使われる[12] |

# (復習) REPORTボディの構造

- sgx_report_body_tの構造（2/2）

| メンバ | 説明 |
| --- | --- |
| **sgx_prod_id_t** isv_prod_id | Enclave設定XMLで設定するISVのProd ID値 |
| **sgx_isv_svn_t** isv_svn | Enclave設定XMLで設定するISVのセキュリティ上の番号 |
| **sgx_config_svn_t** config_svn | Enclave設定のSVN。KSS機能を使用する際に使われる[12] |
| **uint8_t** reserved4[42] | 将来のための予約領域。現時点ではオールゼロとする |
| **sgx_isvfamily_id_t** isv_family_id | ISVファミリのID。KSS機能を使用する際に使われる[12] |
| **sgx_report_data_t** report_data | ユーザがREPORTに任意のデータを組み込むための領域 |

# DCAP-RAにおけるQuoteの構造（4/4）

- 署名関連情報のデータ長については自明であるため説明を省略し、署名関連情報の内訳を以下に示す：

| メンバ | 説明 |
|---|---|
| Quote署名 | QuoteヘッダとEnclaveのREPORTに対する、Attestation秘密鍵によるECDSA署名 |
| Attestation公開鍵 | Quote署名に使用したAttestation秘密鍵に対応するAttestation公開鍵 |
| QE3 REPORT | そのQuote構造体の生成に使用されたQE3のQE3 REPORT。プロビジョニング時にシーリングされストアされたものをアンシーリングして格納する |
| AK Cert | そのQuote構造体の生成に使用されたQE3のQE3 REPORTに対する、PCK秘密鍵によるAK Cert。プロビジョニング時にシーリングされストアされたものをアンシーリングして格納する |
| QE3認証データ | 「AK Certの生成」で説明した、QE3が指定する追加情報。Intel公式のQE3では、ダミーのバイナリ列をプレースホルダとして使用している |
| QE3証明情報 | QuoteやQE3（Attestationキー）の信頼性を証明するための付属情報。証明情報タイプと証明情報サイズ、そして証明情報本体から構成される。デフォルトでは証明情報タイプは5となり、その場合証明情報本体はPCK Certチェーンとなる |

# Quoteの構築（1/4）

- Quoteヘッダのメンバは全てQE3内で用意できるため、**Quoteヘッダ**は**QE3内でセットアップ**する

- 先程LAの完了処理を行い、検証対象EnclaveのREPORTのチェックは済んでいるため、そのままREPORTボディを取り出して**Quote構造体内に格納**する

# Quoteの構築（2/4）

- 署名関連情報について、手始めに本命であるQuote署名の生成から始める

- Quote署名は、前述の署名形式の下、**QuoteヘッダとREPORT**の連結に対して**Attestation秘密鍵で署名**する事で生成し格納する

- 同時に、プロビジョニングで用意された**Attestation公開鍵**、**QE3 REPORT**、**AK Cert**をアンシーリングし、そのままQuote構造体に格納する
  - 前述の通り、厳密には上記のAttestation秘密鍵とこの3つの値は同一のシーリングデータに内包されているため、最初にアンシーリングして適宜使用する形になる

# Quoteの構築（3/4）

- **QE3認証データ**についても**QE3が指定するデータ**であるため、そのままQE3内でQuote構造体にコピーする
  - 前述の通り、Intel製QE3ではダミーのバイナリ列（0x00～0x1Fまでの32バイト）をQE3認証データとしている

- 残るQE3証明情報であるが、これは**QE3**（に強く紐づく要素である**Attestationキー**）の**信頼性を証明するため**のものである
  - 構成としては、証明情報のタイプを示す値（証明情報タイプ）、証明情報サイズ、そして証明情報本体から成る

- Attestationキーの信頼性はAK Certを検証する事で行えるため、これは即ち原則として**PCK Cert**である
  - PCK Cert内のPCK公開鍵でAK Certを検証する

# Quoteの構築（4/4）

- PCK Certはコラテラルであるため、PCSやPCCSといった**コラテラルサービスから取得**する必要がある

- PCK Certに限らず、コラテラルをコラテラルサービスから取得するには、**Intel QPL**（Quote Provider Library）が呼び出される
  - 手順に従いDCAPライブラリをインストールしていればデフォルトで使用可能なライブラリである

- 取得したPCK Certはそのまま**QE3証明情報として**Quoteに格納する
  - PCK Certの場合、証明情報タイプは5となる
  - QPLが何らかの理由（/etc/sgx_default_qcnl.confの記述ミス等）でPCK Certのフェッチに失敗した場合は、代わりに暗号化済みPPIDが証明情報として格納され、タイプは3となる

# Quote署名対象範囲についての議論（1/2）

- ところで、**Quote署名**は**QuoteヘッダとREPORTに対してのみ**付与されており、**それ以外の要素**には**かかっていない**
  - それ以外の要素：Attestation公開鍵、QE3 REPORT、AK Cert、QE3認証データ、QE3証明情報

- 一見それらについてはQuote署名による**改竄耐性がかかっていない**ように見えるかも知れないが、実際は**上手い具合に改竄検知できる**仕組みとなっている

# Quote署名対象範囲についての議論（2/2）

- まず、QE3証明情報であるPCK Certを正規のものから
  すり替えると、以下の3つのいずれかとなり**瞬時に検知可能**
  - IntelのルートCA証明書による検証に失敗する
  - 全く関係ないTCBレベルについてのPCK Certである
  - 失効している

- また、**Quoteの署名**自体は**QE3内で安全に行われるため**、
  **AK Cert**及びその署名対象（To Be Signed; TBS）である
  **QE3 REPORT**は、改竄されると**Quote検証で失敗**する
  - つまり、QuoteとPCK Certの両側からAK Certはチェーンに
    繋ぎ止められており、改竄発生時には検知できる

- Attestation公開鍵やQE3認証データは、REPORTの**Report Data**
  にハッシュが格納されているので、これも改竄検知可能

# Quote検証ロジック詳説

# Quote検証コードの注意点

- DCAPの公式ライブラリ（DCAP 1.20時点）では、**Enclave外で動作するロジック**や、**全くQvEを用いないQvLによる検証**に、**qve.cpp（QvE用のEnclaveコード）を流用**している

- つまり、qve.cppの関数が呼ばれていても、それが**必ずしもECALLであるとは限らない**
  - 構成及び可読性の双方の面で**極めて行儀が悪い実装方法**である

- よって、QvE使用時に実際にQvEへのECALLが発生しているかは、Edger8rにより呼び出し引数（Enclave ID、本来の戻り値）が追加されている、**ECALL独特の形になっているかで判別**する

# コラテラルの取得（1/4）

- 手始めに、指定されたQvEロードポリシーに応じた方法で**QvEをロード**（起動）する
  - QvLを用いる場合は不要

- その後、**Quoteの検証に必要なコラテラル**として、**PCK CRL**、**TCB Info**、**QE3同一性情報**、**ルートCA CRL**を**コラテラルサービスから取得**する
  - PCK CRL、TCB Info、QE3同一性情報は対応する発行者証明書チェーン（コラテラルの署名に対する証明書と、IntelルートCA証明書の2段階から構成されるチェーン）も取得する
  - ちなみに、PCK CRLの発行者証明書チェーンは**取得はするが未使用**である
  - 取得方法の詳細は次ページ以降で説明

# コラテラルの取得（2/4）

- コラテラルの取得にはPCK Certに含まれる値が必要なため、まずは**Quote**から**PCK Certチェーンを抽出**する
  - 前述の通り、PCK Certは検証対象マシンのセキュリティレベルを体現するものであるため、その中に含まれる値によるクエリは検証対象マシンに関わるコラテラルをフェッチするために使える
- PCK Certチェーンからは**PCK Cert**、**中間CA証明書**、**ルートCA証明書**をパースし取り出しておく
- その後、PCK CertのX.509拡張領域から**FMSPC値**を、X.509発行者情報から**CA情報**を取り出す
  - CA情報は”processor”か”platform”のいずれかの文字列であり、マルチソケット環境か否かで決まると推測される

# コラテラルの取得（3/4）

- **PCK CRL**とその発行者チェーンは、**FMSPC値とCA情報の双方**をクエリとしてコラテラルサービスから取得する
- **TCB Info**とその発行者チェーンは、**FMSPC値のみ**をクエリとしてコラテラルサービスからの取得を行う
- **TCB Info**：そのプラットフォームでサポートされる**全てのCPUSVNとPCESVNをリスト化し保持**している構造体
  - **PCK Cert内**のこの**2つのSVN**を**TCB Infoのリストと照合**する事により、**そのマシンのセキュリティレベルを確認**する
  - **SVNが一致したエントリ**内の**TCBステータス**（OKやOUT_OF_DATEのようなQuote検証結果ステータス）や**アドバイザリID**が、そのまま**検証対象マシン**の**セキュリティ情報**（信頼可能性）となる

# コラテラルの取得（4/4）

- **QE3同一性情報**については、その時点で最新のIntel製QE3の同一性情報を持ってくれば良いため、**クエリとしては特に指定せず**コラテラルサービスから単にフェッチする

- **QE3同一性情報**：文字通りIntel製QE3についての同一性情報。Quote内の**QE3 REPORTと比較**する事により、Quote生成において**真にIntel製QE3が使われている**事を**確認**できる
  - **Intel製以外のQE3**（例：自作QE3）**を用いる場合**は**この情報のフェッチや使用は不要**

- 最後に、**ルートCA CRL**については**コラテラルのバージョン**に応じてクエリを構成しコラテラルサービスから取得する

# コラテラルのバージョンチェック

- 各種コラテラルの取得後は、コラテラルのバージョンについてチェックし、後続の処理のために各種変数を初期化する

- QvEを用いる場合は、ここのバージョンチェック開始のタイミングで**ECALL**（sgx_qve_verify_quote()）が**発生**する

- QvLの場合はsgx_qvl_verify_quote()が呼び出されるが、これは中でマクロ付けにより無理矢理sgx_qve_verify_quote()が呼び出されるようになっている
  - 前述の通りの、**QvE不使用時**にも**QvEのコードを使う非常に汚い実装**

# PCK Certの署名検証（1/3）

- 次に、**PCK Cert自体の信頼性**が確かであり、そして紐づく**PCKが失効していない**事を、より**上位の証明書**（中間、ルート）と**PCK CRL**を用いて検証する

- まず、ルートCA証明書はルートCA証明書を用いて**自己検証**する
  - この「ルートCA証明書検証のためのルートCA証明書」には、予めPCK Certチェーンから抽出したルートCA証明書を用いている
  - 大元はIntel PCSから配布されているものであるため、わざわざOSにインストールしたりする必要はSGXの脅威モデル的に無い、と判断しての措置であると推測される

- ルートCA証明書の公開鍵はQvEにハードコーディングされているため、**チェーンの偽造の心配はない**

# PCK Certの署名検証（2/3）

- その後、**中間CA証明書**は**ルートCA証明書**を用いて検証し、**PCK Cert**は**中間CA証明書**を用いて検証する

- CRLについては、**ルートCA CRL**は**ルートCA証明書**を用いて、**PCK CRL**については**中間CA証明書**を用いて、CRLに付与された署名の検証を行う
  - **PCK CRL**については、コード内では「**中間CA CRL**」とも表現される

- そして、**中間CA証明書**が**ルートCA CRL**により**失効**されていないかと、**PCK Cert**が**PCK CRL**（中間CA CRL）により**失効**されていないかを検証する
  - 失効確認は、CA証明書のシリアルナンバーがCRLに含まれていないか（含まれていれば失効済み）を確認する事により行う

# PCK Certの署名検証（3/3）

- ちなみにPCK Certチェーンにおいては、証明書を**1つ上位のCAが発行したCRL**で**失効**させている

- これは、DCAP-RAではその**証明書の対象が危殆化**するだけでなく、脆弱性等により**主体そのものが破綻**する可能性があるために、意図的に取られている構成であると考えられる
  - 例：**PCK秘密鍵が危殆化**するだけでなく、**PCEやマシンが危殆化**する可能性があるため、PCEではなく**Intel**（の中間CA）が**失効処理を行う**必要がある

- 最後に、各種証明書（ルート・中間・PCK Cert）と、2つのCRLが期限切れを起こしていないかを確認する
  - 前述の通り、期限切れを起こしていても致命的なエラーではない。以下、全てのコラテラルの期限切れについて同様

# TCB Infoの署名検証（1/2）

- まず、TCB Info発行者証明書チェーンから**TCB署名証明書**と**ルートCA証明書**をパースし取り出す
  - **TCB署名証明書**：TCB Infoに付与された署名の検証に使用できる、中間CA証明書的な立ち位置の証明書

- 次に、**TCB署名証明書**がルートCA CRLによって**失効されていない**事を**確認**する

- **ルートCA証明書**は**PCK Certチェーン由来**の**ルートCA証明書**で**検証**し、**TCB署名証明書**は検証した**TCB Info発行者証明書チェーン由来**の**ルートCA証明書**で**検証**する

# TCB Infoの署名検証（2/2）

- そして、**TCB Infoの署名**を**TCB署名証明書**内の主体者**公開鍵**で**検証**する

- 最後に、TCB Info発行者チェーン由来のルートCA証明書、TCB署名証明書、ルートCA CRL、TCB Infoが期限切れを起こしていないかを確認する

# QE3同一性情報の署名検証

- QE3同一性情報の署名検証方法は、前述の**TCB Infoの場合とほぼ同様**である
- まずはQE3同一性情報をパースし、EnclaveIdentityV2型という専用のオブジェクトにパースする
- その後、TCB Infoの場合と同様にQE3同一性情報発行者チェーンの署名や失効を検証し、QE3同一性情報の署名についても検証する
- 各種期限切れチェックを行うのも同様

# Quote検証本処理開始 – Quoteフォーマットチェック

- コラテラルの署名等の検証完了後は、ようやくQuote自体の検証の本処理に突入する
- まず、受け取ったQuoteを専用のオブジェクトにパースする
  - AK CertやQuote内PCK Cert等も全てここで取り出す
- そして、Quoteの各要素の立て付け（サイズ等）やQuoteのバージョン、QuoteのTEEタイプ（SGX用かTDX用か）といった**フォーマット**や**メタデータ**の**チェック**を行う
- Quote内の**QE3証明情報タイプのチェック**もここで行う
  - 具体的には、現行ではタイプが1～5の範囲に収まっている必要がある

# コラテラルのパース（1/2）

- Quote検証に使用する**各種コラテラル**（PCK CRL、TCB Info、QE3同一性情報、PCK Cert）を改めて**専用のオブジェクトにパース**する
  - ルートCA CRLは既に各種コラテラルの失効確認をしているので用済みであり、Quote検証本処理には渡されない

- PCK CertのX.509拡張領域から、検証対象マシンの**PPID**、**TCB Info**、**PCEID**、**FMSPC**、**SGXタイプ**を抽出する
  - **PCK Cert内のTCB Info**は「**検証対象マシンのセキュリティレベル**についての**単一のTCB情報**」である
  - **コラテラルの方のTCB Info**は、前述の通り**そのマシンで取り得る全ての TCB Infoのリスト**である

# コラテラルのパース（2/2）

- ちなみに、SGXタイプとは**レガシーSGXかScalable-SGXか**を表す値である

- 興味深い事に、DCAP 1.19の時点で「**ScalableWithIntegrity**」というタイプが用意されている
  - 文字通り、完全性を伴うScalable-SGXである事になる

- Scalable-SGXは、**EPCサイズの劇的な拡大を実現**した一方で**メモリ完全性保証能力を失っている**ため[8][9]、将来的に**この欠点を解決**したCPU等が**リリースされる伏線**であるとも取れる

# PCK CertとTCB Infoのチェック（1/2）

- 念押しの意味で、もう一度**PCK Certと**TCB Info**コラテラル**（コラテラルの方のTCB Info）の**チェック**を行う

- PCK Certについては、発行者と主体者の整合性を確認し、またPCK CRLによりそのPCK Certが失効していない事を確認する

- TCB Infoについては、TCB Infoのバージョンを確認し、そのTCB InfoがSGXとTDXのどちら用であるかを確認する
  - 今回は当然SGXのRAであるため、TDX用であればエラーとする

# PCK CertとTCB Infoのチェック（2/2）

- FMSPC値とPCEIDについて、**PCK Cert内のFMSPC値**で**TCB Infoをリクエスト**しており、かつ**PCEID**はその**PCK Certに対応するPCE**に**固有な値**である

- よって、PCK Cert内のこの2つの値とTCB Info内のそれらは**必ず一致する必要がある**ため、その一致確認を行う

# AK Certの検証（1/2）

- いよいよQuote検証本処理の核心に突入していく

- これまでの処理で**PCK Certチェーンの信頼性を確認**する事に**成功**しているため、次はPCK Certよりも**1段下位**に**位置**する**AK Cert**の検証を行う

- AK Certの**署名**、**署名対象**である**QE3 REPORT**、そしてPCK Cert内の**PCK公開鍵**を取り出し、ECDSA署名検証関数で**AK Certの署名を検証**する
  - AK Certの署名はPCK秘密鍵でQE3 REPORTについて打たれているため、対応するPCK公開鍵で検証するという、一般的な電子署名の議論

# AK Certの検証（2/2）

- 前述の通り、AK Certの署名対象であるQE3 REPORTは、**Report Data領域**に**Attestation公開鍵**と**QE3認証データ**の**連結に対するハッシュ値**を**保持**している

- よって、**Quoteに格納**されている**Attestation公開鍵**と**QE3認証データ**を取り出し、**その連結に対するハッシュ値**を上記**Report Data内のハッシュ値**と**照合**する
  - 一致すれば改竄が発生していない事になるので合格

# QE3の同一性検証（1/7）

- AK Certの検証の成功により、AK Certの署名対象の**QE3 REPORT**が**信頼可能な情報**である事が**この時点で証明**されている

- つまり、**QE3の同一性**（MRSIGNER等）を**忠実に証明するデータ**として**使用可能**である事になる
  - QE3内で生成されたQE3についてのREPORTであり、かつAK Certの検証で改竄が無い事を確認済みであるため

- よって、Intel製QE3を用いている場合は、このQE3 REPORTとコラテラルのQE3同一性情報を比較し、**真にIntel製QE3が使用されている事を確認**してクライアントに示せる
  - 逆に、Intel製以外のQE3を用いている場合はこの検証処理は不要

# QE3の同一性検証（2/7）

- **QE3 REPORTとQE3同一性情報**それぞれから取り出した以下の**同一性情報を比較**し、Intel製QE3である事を確かめる
  - **Misc Select**：現在ではまだ予約用変数である sgx_misc_select_t 。QE3同一性情報から抽出したMisc Select用マスクビットをQE3 REPORT由来のMisc Selectに適用してから比較を行う
  - **属性情報**：Enclaveが使用する拡張機能等を示す sgx_attributes_t 。これもQE3同一性情報由来の属性情報マスクを上記と同様に適用してから比較する
  - **MRSIGNER**
  - **ISV Prod ID**
  - **ISVSVN（QE3SVN）**：比較方法が特殊であるため**次ページ以降で説明**

# QE3の同一性検証（3/7）

- コラテラルのQE3同一性情報には、想定する**QE3SVN**の**リスト**と、**各QE3SVNについて**の**タイムスタンプ**や**TCBステータス**が同梱されている
  - QE3SVN、タイムスタンプ、TCBステータスの3つをまとめて**TCBレベル**と表現する

- 許容される範囲内の**全ての取り得るQE3SVN**についての**エントリ**が**用意されているわけではない**が、**QE3SVN**は、**期待する値よりも大きければ良い**数値である
  - これはQE3SVNに限らずISVSVN等SVN全般に言える話である

- よって、QE3SVNが**QE3 REPORT由来のそれ以下**であり、かつ**そのようなエントリ**の内**最大のQE3SVNのTCBレベル**を取得すれば良い

# QE3の同一性検証（4/7）

- TCBレベルの取得方法について：例えば、コラテラルのQE3同一性情報内に**1, 3, 5, 7**の**4つのQE3SVN**についてのTCBエントリが存在するとする

- QE3 REPORT内のQE3SVNが**6**である場合、**6以下かつその中で最大**であるQE3SVN=**5**についての**TCBレベルエントリを取得**すれば良い

- TCBレベルエントリを抽出後は、そのエントリ（つまり、ひいては**QE3**）に紐づく**TCBステータスを抽出**する

# QE3の同一性検証（5/7）

- TCBステータスは、以下のようにEPID-RAでも見覚えのあるような品揃えとなっている：

```
enum class TcbStatus {
        UpToDate,
        ConfigurationNeeded,
        OutOfDate,
        OutOfDateConfigurationNeeded,
        Revoked,
};
```

- UpToDate（最新で信頼可能）である場合は、QE3検証結果をSTATUS_OKとする
  - この場合は、クライアントはQE3を完全に信頼可能

# QE3の同一性検証（6/7）

- UpToDateとRevoked以外の場合は、検証結果をSTATUS_SGX_ENCLAVE_REPORT_ISVSVN_OUT_OF_DATEとする

- Revokedであるか、そもそもその**QE3SVN以下**の**TCBレベルエントリ**が**存在しない**場合は**失効**している事になるため、STATUS_SGX_ENCLAVE_REPORT_ISVSVN_REVOKEDとする

# QE3の同一性検証（7/7）

- QE3SVN以外の同一性検証で**不一致**が発生している場合は、ただちに対応するエラーステータスと共に**リターン**する

- 一方、OUT_OF_DATEであったり失効している場合については、QE3が古くはなっているがそれを信頼するかは**クライアントの判断次第**である
  - よって、Quote検証処理自体は**そのまま続行**する

- QE3同一性検証を行うかは、検証関数にコラテラルのQE3同一性情報からパースされたEnclaveIdentityV2オブジェクトが渡されているかによって決定される

# Quote署名の検証

- AK Certまでの署名は既に検証済みであるため、いよいよ**Quote署名の検証**を行う

- AK Certの検証によりQuote内のAttestation公開鍵が正当である事が確認済みであるため、単に**Attestation公開鍵**により**Quote署名を検証**すれば良い

- この時点で、IntelルートCA証明書からQuoteまでのDCAP-RA**トラストチェーン全体の検証に成功**しているため、以降その**Quote**を検証対処のマシンとEnclaveについての**信頼可能な同一性情報**として**取り扱う事**ができる

# 検証対象のTCBレベルのチェック（1/3）

- **PCK Cert**から取り出した**検証対象**についての**TCBレベル**を、**TCB Infoコラテラル内**の適切な**TCBレベルエントリと比較**する事で、**検証対象のTCBレベルを確かめる**

- TCB Infoコラテラルに含まれるTCBレベルエントリの内、**TCBコンポネントSVN**と**PCESVN**が、全て**PCK Cert内の TCB Info**以下かつその中で最大のものを取り出す
  - QE3同一性情報における抽出処理と同様である

- **TCBコンポネントSVN**は16個のSGXコンポネントそれぞれに対するSVNであるが、この16個の集合が**CPUSVN**である

# 検証対象のTCBレベルのチェック（2/3）

- 例えば、TCBコンポネントSVNとPCESVNが**全て1**（1, 1, ..., 1）、**全て3**、**全て5**、**全て7**のTCBレベルエントリがTCB Infoコラテラルに含まれているとする

- PCK Certから抽出したTCBレベルのSVNが**全て6**である場合は、6以下かつその中で最大である、**全て5**の**TCBレベルを抽出**して**比較処理に使用**する

- QE3同一性情報と異なり、PCK Cert内のFMSPC値でTCB Infoコラテラルをフェッチしているため、**上記条件を満たすTCBレベル**エントリは**必ず含まれている**必要がある
  - 含まれていない場合は直ちにエラーとしてリターンする

# 検証対象のTCBレベルのチェック（3/3）

- 条件を満たすTCBレベルエントリを抽出したら、それに紐づいている**TCBステータス**を**取り出し**、それを検証対象マシンの検証結果（**Quote検証結果ステータス**）とする

- Quote検証結果ステータスは、**EPID-RA**におけるそれらと**ほぼ同じ立て付け**となっている：
  - STATUS_TCB_OUT_OF_DATE
  - STATUS_TCB_REVOKED
  - STATUS_TCB_CONFIGURATION_NEEDED
  - STATUS_TCB_CONFIGURATION_AND_SW_HARDENING_NEEDED
  - STATUS_OK
  - STATUS_TCB_SW_HARDENING_NEEDED
  - STATUS_TCB_OUT_OF_DATE_CONFIGURATION_NEEDED

# Quote検証結果補足情報のセットアップ

- これにてQuote検証処理が完了したため、Quote検証結果についての参考情報として**補足情報**を**セットアップ**する：

| メンバ | 説明 |
|---|---|
| **uint32_t** version | 補足情報のバージョン。現在では下記のメジャー/マイナーバージョンで表現する仕様となっているため、後方互換性のために用意されているメンバである |
| **uint16_t** major_version | 補足情報のメジャーバージョン |
| **uint16_t** minor_version | 補足情報のマイナーバージョン |
| **time_t** earliest_issue_date | 全てのコラテラルの発行日時の内、一番古い日時がここに格納される |
| **time_t** latest_issue_date | 全てのコラテラルの発行日時の内、一番新しい日時がここに格納される |
| **time_t** earliest_expiration_data | 全てのコラテラルの期限切れ日時の内、一番早い日時がここに格納される |

# Quote検証結果補足情報のセットアップ

- 補足情報（sgx_ql_qv_supplemental_t）の構造続き（2/4）：

| メンバ | 説明 |
|---|---|
| **time_t** tcb_level_date_tag | SGX脆弱性（セキュリティアドバイザリ）に対する軽減策の内、このメンバが示す日時までにリリースされたもの全てが検証対象マシンに適用されている事を示す。 |
| **uint32_t** pck_crl_num | PCK CRLのCRL番号<br>（各CRLに対し一意に割り当てられるID） |
| **uint32_t** root_ca_crl_num | ルートCA CRLのCRL番号 |
| **uint32_t** tcb_eval_dataset_num | 検証に使用されたコラテラルのバージョンを簡易的に把握するための値[10]。上記のCRL番号を<br>用いても同様の把握ができるため、どちらか任意の方法を選択する事ができる |
| **uint8_t** root_key_id[48] | IntelルートCA証明書の公開鍵に対する<br>SHA384ハッシュ値 |
| **sgx_key_128bit_t** pck_ppid | 検証対象プラットフォームのPPID。<br>プラットフォーム同一性の確認に利用できる |

# Quote検証結果補足情報のセットアップ

- 補足情報（sgx_ql_qv_supplemental_t）の構造続き（3/4）：

| メンバ | 説明 |
|---|---|
| **sgx_cpu_svn_t** tcb_cpusvn | 検証対象マシン（に紐づくPCK Cert内）のCPUSVN |
| **sgx_isv_svn_t** tcb_pce_isvsvn | 検証対象マシン（に紐づくPCK Cert内）のPCEのISVSVN（PCESVN） |
| **uint16_t** pce_id | 検証対象マシンのPCEID |
| **uint8_t** sgx_type | 検証対象マシンのSGXタイプ |
| **uint8_t** platform_instance_id[16] | マルチソケットプラットフォームに対し一意に割り当てられたID。PPIDに似ているが、こちらはプロビジョニングに対するIDではなく、プラットフォーム自体に対するIDとして機能する、マルチソケット環境専用の値のようである[11] |
| **pck_cert_flag_enum_t** dynamic_platform | 原文直訳：「SGX Registration BackendへのPackage Addコールにより、追加パッケージでプラットフォームを拡張できるかどうかを示す」。プロビジョニング後にさらにCPUパッケージを追加できるマシンであるかのフラグであると推測される |

# Quote検証結果補足情報のセットアップ

- 補足情報（sgx_ql_qv_supplemental_t）の構造続き（4/4）：

| メンバ | 説明 |
| --- | --- |
| **pck_cert_flag_enum_t** cached_keys | 原文直訳：「SGX Registration Backendによってプラットフォームルート鍵がキャッシュされるかどうかを示す」。<br>これは補足で「Direct Registration vs Indirect Registration」とあるので、マルチソケットプラットフォームの登録方法についてのフラグであると推測される[12] |
| **pck_cert_flag_enum_t** smt_enabled | ハイパースレッドが有効化されているかのフラグ。ハイパースレッドはSGXへの攻撃に悪用される事が非常に多いため、基本的に無効化されている事が望まれる |
| **char** sa_list[320]; | 検証対象マシンが抱えるSGX関連の脆弱性に対するアドバイザリIDの一覧。各IDはカンマ区切りされており、リストはヌル終端されている。例えば、Downfall（GDS）脆弱性を抱えている場合は、INTEL-SA-00828というエントリが入っている。 |

# QvE REPORTの作成

- QvEを用いて検証している場合は、受け渡されている**QvE REPORT情報**に含まれている、**検証対象Enclave**の**Target Info**を**入力**とした、そのQvEについてのREPORT（**QvE REPORT**）を生成する

- QvE REPORTのReport Data領域には、以下の全てのデータに対するSHA256ハッシュ値が格納される：
  - QvE REPORT情報内のnonce
  - Quote
  - タイムスタンプ
  - コラテラル期限切れステータス
  - Quote検証結果
  - 補足情報

# Quote検証処理の完了

- これにてQuote検証処理は完了であるため、再度の言及となるが、Quote検証サーバは例えば以下のデータをクライアントに返信する：
  - 検証したQuote
  - Quote検証結果ステータス
  - コラテラル期限切れステータス
  - 補足情報
  - **QvE REPORT（後ほど議論）**

# セッション共通鍵の生成

- 無事**RAを受理**したら、RPは**Attesterとの暗号通信**（**128bit AES/GCM暗号化**）に使用する、**セッション共通鍵**の**SK**と**MK**を生成する（かつてのEPID-RAの作法に準じた手順）

- SKとMKは、それぞれ以下のバイト列に対しKDKを鍵として生成した128bit AES/CMAC値である：
  - SK：バイト列「¥x01SK¥x00¥x80¥x00」
  - MK：バイト列「¥x01MK¥x00¥x80¥x00」

- Attesterでも同様にしてEnclave内でSKとMKを算出する

# SPとISV Enclaveとの暗号通信

- RA成立後に安全にEnclaveと通信するには、RPは**SKかMK**で**送りたいデータ**を**暗号化**し、**RAコンテキスト**や**暗号など**を**Attesterに送信**し、Enclave内に読み込み**Enclave内で復号**させる
  - デフォルトで提供されているSGXAPIの都合上、暗号方式は**128bit AES/GCM**となる
  - 基本的に**SKやMK**は**Enclave外に出る事はない**。OCALL等で無理矢理出す事も出来るが、そういった**悪性のEnclave**は、**RA前**にMRENCLAVEを得る時点で**目視等で確認**しておく

# AES/GCMの特徴

- 平文と暗号文の**バイト長が同じ**である
- 暗号化と復号には**初期化ベクトル**（IV）を用いる
  - **IVは公開情報**であり、普通**12バイト**である事が多い（NISTによる推奨）
- 暗号化すると、暗号文の他に**GCMタグ**（メッセージ認証符号）も出力され、復号時にはこのタグも渡す必要がある
  - タグは**16バイト**であり、**公開情報**である
- 必要に応じて追加認証データ（AAD）を同梱する事も出来る

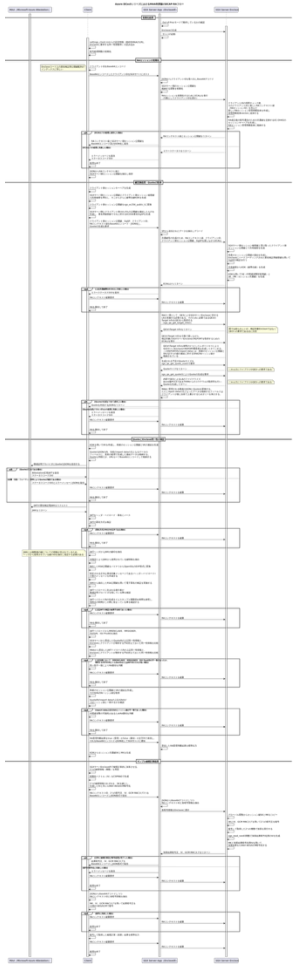
# AES/GCM暗号処理

- Enclave内においては、
  AES/GCM暗号化は**sgx_rijndael128GCM_encrypt**関数を、
  復号には**sgx_rijndael128GCM_decrypt**を使用する

- RPにおいては、例えば**OpenSSL**の適切な関数を用いて
  AES/GCM暗号化及び復号を行うと良い

# RAの詳細フロー全体のシーケンス図

- ここまでで説明したRAのフローを厳密に**シーケンス図**に起こすと
  **以下のようになる**



# 実践・RAの実装

- それでは、**ここまでの内容**を踏まえ、**RAを実行するRPと Attesterのプログラムを実装**しましょう！

初学者に実装させるには
RAはあまりにも非人道的

# Humane-RAFW-MAA

# Humane-RAFW-MAA（1/2）

- EPID-RAとは異なり、DCAP-RAは公式サンプルコードがまだマシではあるが、依然ゴチャついている感は否めない[15]

- そこで、本来であればDCAP-RA版Humane-RAFWを講師側で実装し提供すべきである

- しかし、諸事情により実装が追いついておらず、代わりに**MAA（Microsoft Azure Attestation）**をQuote検証機関として用いる、**Humane-RAFW-MAA**を提供している
  https://github.com/acompany-develop/Humane-RAFW-MAA

# Humane-RAFW-MAA（2/2）

- **MAA**：Azureが提供するQuote検証サービス。**構成証明プロバイダ**と呼ばれるサービスのインスタンス（無料）を立て、そのURLに**Quote**を含むJWTを**送信**する事で利用可能

- **検証作業**を**Azureに丸投げできる**のは大きな**メリット**である

- 反面、Azureを完全に信頼する必要があったり、Quote検証結果が例えばSW_HARDENING_NEEDEDでも**OK**（http status 200）として**返してくる**等、**信頼性の観点での懸念はある**
  - 一応Quoteが改竄されているとエラー400を返してくる。ただ、それ以上のQuote検証をMAAが正確に行っている事は、外側からでは一切確認や保証する事ができない

# QvEに対するLAについての議論

# 検証対象EnclaveによるQvEに対するLA（1/4）

- **QvE**が**検証対象Enclave**と**同一マシン上で動作**している場合に限り、検証対象Enclaveは**QvEに対してLAを実行**し、その**正当性を検証**する事ができる
  - APIはsgx_tvl_verify_qve_report_and_identity()

- QvE REPORTのReport Dataに含まれる前述のハッシュ値についても、検証対象Enclaveが受け取った各種情報を用いてその正しさを検証する

# 検証対象EnclaveによるQvEに対するLA（2/4）

- しかし、この検証処理は根本的に無意味であると推測される

- 例えば悪性の検証対象Enclaveが、**任意の同一性情報**に対する**REPORT構造体**を、適当なバイナリ列をレポートキーとして**偽造**するとする

- さらに、**適当なAttestationキー役のECDSA鍵**で**偽造REPORT**に**署名**し、**偽造Quoteを生成**したとする

# 検証対象EnclaveによるQvEに対するLA（3/4）

- 当然、**QvE**はこの**偽造Quote**に対して**不正である**との**判定**を下し、そのQuote検証結果を**QvE外にリターン**する
  - 偽造はAK Cert等の検証で検知できる

- しかし、QvEが検証対象Enclaveと同一マシン上で動作している場合、SGXの脅威モデル上そのマシンは**特権攻撃者に掌握されている**可能性もあるため、その**Quote検証結果は改竄され得る**

- よって、いくら検証対象EnclaveがQvEの同一性を確かめようが、そもそも**Quote検証結果自体が改竄**されるため、**根本的に無意味**

# 検証対象EnclaveによるQvEに対するLA（4/4）

- クライアントがこの攻撃を検知するには、再度**クライアント**が自前でコラテラルサービスから**各種コラテラルをフェッチ**し、AK CertでQuote署名に使われた**偽造Attestationキーを検知**する必要がある

- しかし、**Quote検証**を**委託したい**がために**QvEに任せていた**にも関わらず、**再度それと同じ検証処理を自前で行う**のであれば、そもそも**検証対象マシン上**での**検証処理**が**完全に無駄**

- 結局の所、検証対象マシンでQvEを動作させるのは**危険**であり、**検証対象によるQvEに対するLA**も**無意味**である
  - これを問う質問へのIntelからの返答は2024/4/21現在無し[12]

# Intel Trust Authority

# Intel Trust Authority（1/2）

- 2023年秋頃[13]にIntelにより正式リリースされた、いわば**DCAP-RA版IAS**とでも表現できるサービス
  - 正式リリースされるまではProject Amberと呼ばれていた[14]

- QvEやQvLを誰が動かすのかといったような、**前述の自由度に伴う3つの議論**を**飛躍的に解決**しやすくなる、期待度の高いQuote検証サービスである

# Intel Trust Authority（2/2）

- ただし、以下の観点から実装面・金銭面での懸念は残る：
  - 記事執筆時点ではリリースされてからまだ**新しい**ため、**不具合に遭遇**した際に**対策に難儀**する可能性がある
  - Trust Authorityを利用しようとした場合、30日間のトライアルから開始する事になる点（=実利用は**有料である可能性が高い**）
  - EPID-RA時代の**IAS**が**完全無料**であったため、Intelはそれに懲りつつIAS時代の損害を回収するべく、**かなりの値段設定**にする可能性も否めない
  - Scalable-SGXはサーバでの SGXを動作させる事が前提となるため、よりエンプラ等での大規模利用モデルが増え、結果としてそれに合わせて**値段も高額**になる可能性がある

# SGX-VaultのSaaS化

# SGX-VaultのSaaS化

- **Humane-RAFW-MAA**をベースとし、これまでに作った**SGX-Vault**を**移植**する事で、SGX-Vaultを**リモートマシンに配置**し**RA後にリモートから利用**できる（SaaSもどき）ように改良する
  - ただし、本ゼミではSPとISVは同一マシン上に存在し、**ローカルホスト通信**で完結する、**擬似的なリモート通信**の形で良い

- Humane-RAFW-MAAによるRAに必要な事前準備は全て**リポジトリのREADMEに記載**してあるので参照しながら進める

- httplibやSimpleJSON、Base64エンコードを用いた送受信のコーディング方法は、Client_App/client_app.cppやServer_App/server_app.cppが参考になる（はず）

# リモート版SGX-Vaultの要件

- 基本的な要件はこれまでに実装したSGX-Vaultと全く同じ

- ただし、**ユーザをクライアント（SP）、SGXマシンをサーバ（ISV）**とし、**リモート**から**各種機能を利用できなければならない**

- 機能の利用に伴う**各通信**は、**セッション共通鍵**である**SK**あるいは**MK**で**保護されていなければならない**

# 本セクションのまとめ

- 特に日本語での解説が深刻に不足しているDCAP-RAについて、DCAPライブラリの詳細な実装をエビデンスとした説明を行った
- Humane-RAFW-MAAを用いる事で、SGX-Vaultをリモートから安全に利用できるように改良した
- DCAP-RAにおける、自由度に伴う3つの議論は深刻であるが、これを手軽に解決する完璧な方法はまだ登場していないと言える

# 参考文献


[1]”Intel SGX – DCAP-RA解体新書”, 自著記事, https://acompany.tech/privacytechlab/intel-sgx-dcap-ra

[2]”インテル® SGX をサポートするインテル® プロセッサー”, Intel, https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/software-guard-extensions-processors.html

[3] Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives, Vinnie Scarlata et al., https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf

[4] ECDSA Attestation解説, 2024/3/26閲覧, https://hatena75.gitbook.io/ecdsa-attestation-details/ecdsa-attestation

[5] sgx_quote_3.h, DCAP 1.20, https://github.com/intel/SGXDataCenterAttestationPrimitives/blob/dcap_1.20_reproducible/QuoteGeneration/quote_wrapper/common/inc/sgx_quote_3.h#L189

[6] Intel SGXのECDSA Attestationにおける検証についての課題とその改善に向けた考察, 矢川 嵩 et al., https://ipsj.ixsq.nii.ac.jp/ej/?action=pages_view_main&active_action=repository_view_main_item_detail&item_id=218810&item_no=1&page_id=13&block_id=8

# 参考文献


[7] sgx_dcap_ql_wrapper.cpp#L368, DCAP 1.20, https://github.com/intel/SGXDataCenterAttestationPrimitives/blob/DCAP_1.20/QuoteGeneration/quote_wrapper/ql/sgx_dcap_ql_wrapper.cpp#L368

[8] Integrity protected access control mechanisms, https://patents.google.com/patent/US20220209933A1/en

[9] Towards TEEs with Large Secure Memory and Integrity Protection Against HW Attacks, Pierre-Louis Aublin et al., https://systex22.github.io/papers/systex22-final15.pdf

[10] Intel® Trust Domain Extensions (Intel® TDX) Data Center Attestation Primitives (DCAP): Quote Library API, Intel, https://download.01.org/intel-sgx/sgx-dcap/1.20/linux/docs/Intel_TDX_DCAP_Quoting_Library_API.pdf

[11] Intel SGX(R) PCK Certificate and CRL Profile Specification, Intel, https://api.trustedservices.intel.com/documents/Intel_SGX_PCK_Certificate_CRL_Spec-1.5.pdf

[12] Question about the reliability of Quote verification results with QvE in ECDSA-RA, https://community.intel.com/t5/Intel-Software-Guard-Extensions/Question-about-the-reliability-of-Quote-verification-results/m-p/1579842

# 参考文献

[13] ゼロトラストを手の届くところに置き、プライベートクラウドセキュリティでパブリッククラウドの柔軟性を実現, Intel, https://www.intel.co.jp/content/www/jp/ja/content-details/788131/put-zero-trust-within-reach-and-get-public-cloud-flexibility-with-private-cloud-security.html?DocID=788131

[14] インテル Trust Authorityサービス契約, Intel, https://www.intel.co.jp/content/www/jp/ja/content-details/784519/intel-trust-authority-services-agreement.html

[15] SGXDataCenterAttestationPrimitives/SampleCode, github, https://github.com/intel/SGXDataCenterAttestationPrimitives/tree/main/SampleCode

[16] Terms of Use for Intel® Software Guard Extensions (SGX) Services and Intel® Trust Domain Extension (TDX) Services, Intel, https://api.portal.trustedservices.intel.com/termsAndConditions